Core C++ 2024

C++ ♥ Python

Alex Dathskovsky

C++

python

About Me:

SPEEDATA

alex.dathskovsky@speedata.io

www.linkedin.com/in/alexdathskovsky

www.cppnext.com

https://www.youtube.com/@cppnext-alexd

Survey Time

CoreC++2024 Survey

SCAN ME

Why Python????

- My survey

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Why Python????

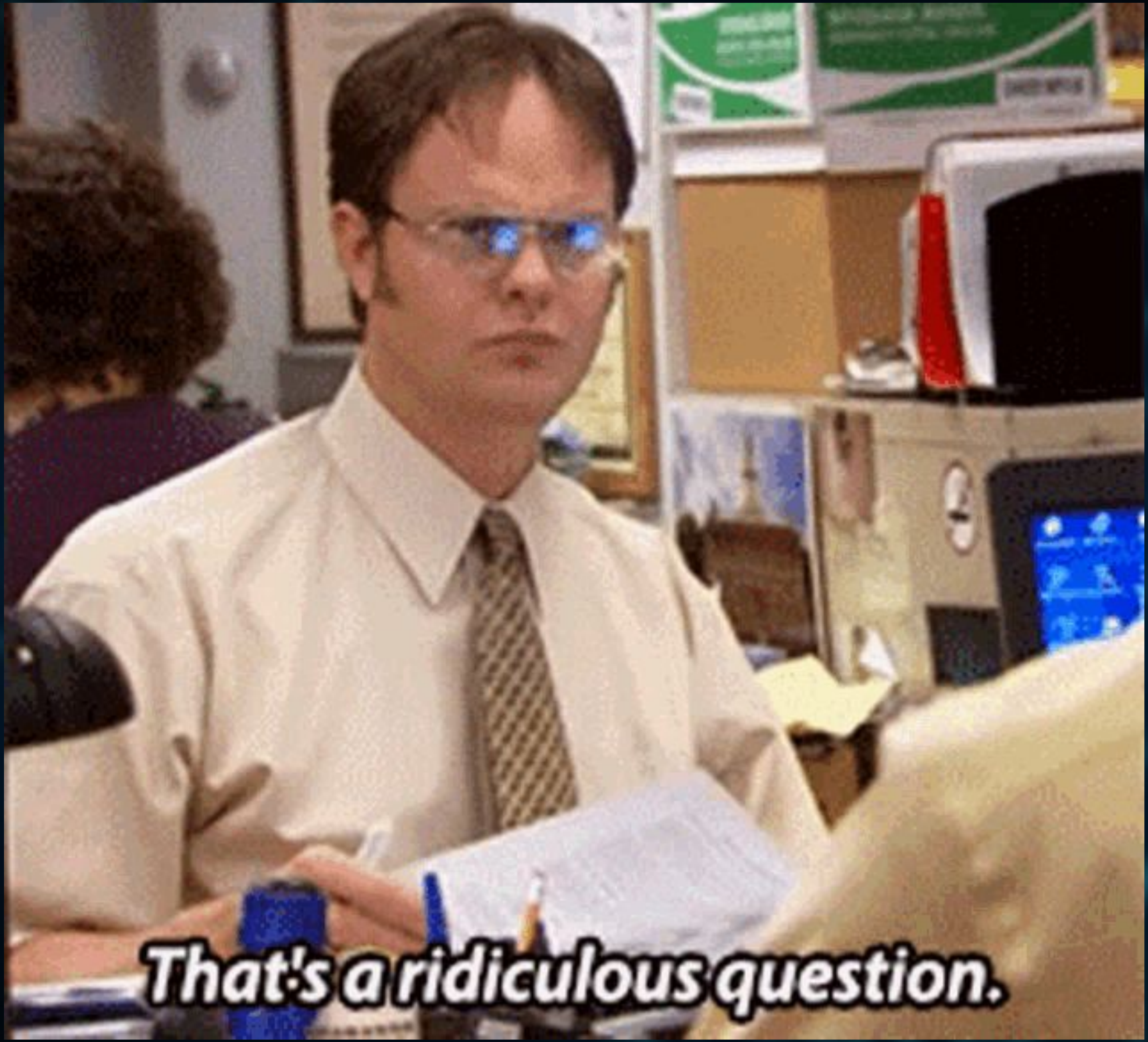
- C++ standard survey

Q16 Besides C++, what programming languages/environments do you use in your current and recent projects? (select all that apply)

Answered: 1,225 Skipped: 40

Python	72.82%	892
C	53.06%	650
JavaScript	27.43%	336
C#	23.76%	291

Why C++ ?



Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Why Python????

Why Python????

- Faster Development
- Simpler Interface/usage
- Interpreted language
  - Easy way to research algorithms

Similarities Between C++ and Python

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Interfaces similar to python in C++

Python:

```
prin
```

```
(1, 2), this is 3
```

```
{1+2}")
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Interfaces similar to python in C++

C++:

```
int main(){
    fmt::pr(1, 2) this is 3    tuple(1,2), 3);
}
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Interfaces similar to python in C++

Python:

```python
for i in [1, 2, 3]:
    print(i)
```

Some Interfaces similar to python in C++

C++:

```
int main(){
    for (auto i : std::array{1, 2, 3}){
        fmt::print("{}\n", i);
    }
}
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Interfaces similar to python in C++

Python:

```python
def foo(a, b):
    return a, b, a+b

a, (1, 2, 3), 4

print(f"{foo(1, 2)}, {a+c}")
```

Some Interfaces similar to python in C++

C++:

```cpp
#include <tuple>
#include <fmt/ranges.h>

auto foo(auto x, auto y){
    return std::tuple{x, y, x+y};
}

int main(){
    auto [a, b, c] = foo(1, 2);
    fmt::print("{}, {}", foo(1, 2), a+c);
}
```

```
(1, 2, 3), 4
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Interfaces similar to python in C++

Python:

```python
def foo(a, b):
    return a, b, a+b

_, _, c = foo(1,2)
```

Some Interfaces similar to python in C++

C++:

```
auto foo(auto a, auto b){
    return std::tuple{a, b, a+b};
}

int main(){
    auto [_, _, c] = foo(1, 2);
}
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Why Can't We Just Use Python?

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

So why can't we just use Python

- Interpreted Language
  - slow

- Easier to make mistakes

Let's Talk About Some Of The Problems

Problems:

```python
def foo(a):
    a + 1

if __name__ == "__main__":
    print(foo(10) + 10)
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Problems:

```python
def foo(a):
    return a + b

if __name__ == "__main__":
    b = 10
    print(foo(10))
```

Problems:

```python
def foo(a, b):
    return a + b

def bar(a, b):
    a + [b]

if __name__ == "__main__":
    print(foo(10, 10))

```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Problems:

```
Traceback (most recent call last):
  File "main.py", line 8, in <module>
    bar(10, 10)
  File "main.py", line 5, in bar
    a + [b]
TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

Problems:

```python
what_is_this = 4294967295

def foo(a, b):

if __name__ == "__main__":
    print(foo(what_is_this, 1))
```

4294967296

Problems:

```python
def foo(a):
    return bin(a), type(bin(a))
```

```
('0b10101101100111', <class 'str'>)
```

```python
print(foo(11111))
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions In Newer Versions

Some Solutions: Python 12 Typing

```python
from typing import List

def foo(a: int, b: int) -> int:
    return a+b

if __name__ == '__main__':
    print(foo(1, 2.1))
```

Expected type 'int', got 'float' instead

test

```python
def foo(a: int,
        b: int) -> int
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: Python 12 Typing

```python
def foo(a: int, b: int) -> int:  1 usage
    a+b

if __name__ == '__main__':
    print(foo( a: 1,  b: 2.1))
```

Expected to return 'int', got no return

builtins

`class int`

int([x]) -> integer int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments are given. If x is a number, return x.__int__(). For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string, bytes, or bytearray instance representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

\`int\` on docs.python.org ↗

Some Solutions: Python 12 Typing

```
from typing import List, Tuple

def foo[T: (int, str)](a: List[T], b: T) -> List[T]:
    a.append(b)
    return a

if __name__ == '__main__':
    print(foo( a: [1, "2"], b: 3))
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: Python 12 Typing

```
from typing import List, Tuple

def foo[T: (int, str)](a: List[T], b: T) -> List[T]:  1 usage
    a.append(b)
    return a

if __name__ == '__main__':
    print(foo( a: [1, "2"],  b: 3.1))
```

Expected type 'int | str' (matched generic type 'T ≤: int | str'), got 'float' instead

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: Python 12 Typing

```python
from typing import List

type U[T] = list[T] | tuple[T, ...]


def foo(a: U) -> List[U]:  2 usages
    return [a]

if __name__ == '__main__':
    print(foo([20.1]))
    print(foo((20.1, 20.2, 20.3)))
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: Linters

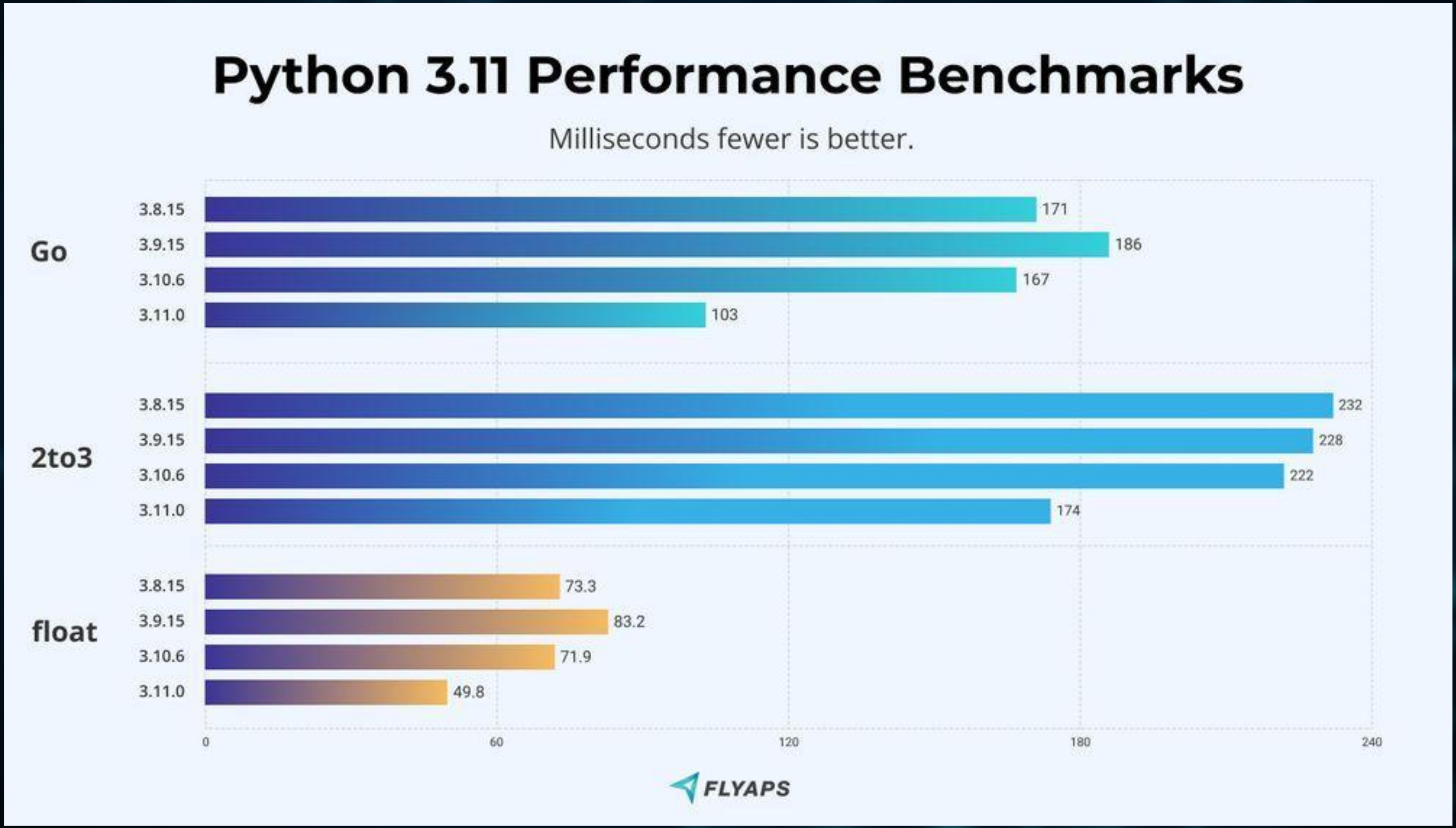
- Flake 8
- MyPy

```
1 from typing import List, Tuple
2
4     return a + ["END!"]
5
6 if __name__ == '__main__':
7     print(foo([20.1]))
```

```
mypy C:\Users\caleb\PycharmProjects\pythonProject\test.py --enable-incomplete-feature=NewGenericSyntax
C:\Users\caleb\PycharmProjects\pythonProject\test.py:4: error: List item 0 has incompatible type "str"; expected "int"  [list-item]
C:\Users\caleb\PycharmProjects\pythonProject\test.py:7: error: Value of type variable "T" of "foo" cannot be "float"  [type-var]
```

Performance

Some Solutions: Python is getting faster



Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: Importance Of Inner Mechanics

```python
import timeit
from random import uniform

rand_val = 100000
a = [uniform( a: 0, rand_val) for i in range(rand_val)]

def func_1():
    s = 0
    for i in a:
        s += i
    return s

def func_2():
    return sum(a)

if __name__ == '__main__':
    print(timeit.timeit(stmt='func_1()', number=rand_val, globals=globals()))
    print(timeit.timeit(stmt='func_2()', number=rand_val, globals=globals()))
```

266.0457221000106

64.64083260000916

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: can we get better?!

```python
import timeit
import numpy as np

rand_val = 100000
a = np.random.randn(rand_val)

def func_3():
    np.sum(a)

if __name__ == '__main__':
    print(timeit.timeit(stmt='func_3()', number=rand_val, globals=globals()))
```

3.3721860999939963

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: what is Numpy

- Abstraction over contiguous multidimensional array
- Export easy to use API for many mathematical functions
- Arrays are not dynamic
- Core implementation is in C and C++ also all types are bound to the language
- Usually constructed for basic C++ types

Some Solutions: Importance of Using Correct Types

```python
import timeit
from random import uniform
import numpy as np

rand_val = 50000
a = [uniform( a: 0, rand_val) for i in range(rand_val)]

def func_2():
    return sum(a)

def func_3():
    return np.sum(a)

if __name__ == '__main__':
    print(timeit.timeit(stmt='func_2()', number=rand_val, globals=globals()))
    print(timeit.timeit(stmt='func_3()', number=rand_val, globals=globals()))
```

```
16.656653799989726
121.12174510001205
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Some Solutions: Importance of Using Correct Types

Some Solutions: Why is Pure python better than Numpy

- We have an overhead
  - Data was created as a pure python list
  - We need to copy the whole data to a ndarry
  - Also data has to be copied from Python memory to C Memory space

Some Solutions: Takeaways

- Always use modules that already define the functionality you need
(no need to write everything by yourself)

- Learn the Module you will use
  - Use functionality as expected
  - Don't abuse the nice API

More Open Source Modules

PyTorch 🔥

What is PyTorch

- Abstraction over contiguous multidimensional array (Sound familiar :))
- Abstraction over AI models and parts
- Easy way to mix and match to build new AI modules
- Core Implementation in C++ AND CUDA
- Very easy to work with CPU and CUDA
- Easy way to train new AI models

Examples: Matrix Calculation on CPU

```
import torch as pt
import timeit

39.814935477000006
a = pt.randn(10 … 0000)
b = pt.arange(5000000).reshape(5, 1000000)
print(timeit.timeit("(a*b)", number=1000, globals=globals()))
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Examples: Matrix Calculation on GPU

```
import torch as pt
import timeit
device = pt.dev
a = pt.randn(10                                    hape(2, 5, 10000000)
b = pt.arange(50000000, device=device).reshape(5, 10000000)
print(timeit.timeit("(a*b)", number=1000, globals=globals()))
```

```
0.04776061099983053
```

Examples: Matrix Calculation on GPU (full round trip)

```
import torch as pt
import timeit
device = pt.devic
a = pt.randn(1000                     e(2, 5, 10000000)
b = pt.arange(50000000, device=device).reshape(5, 10000000)
print(timeit.timeit("(a*b).to('cpu')", number=100, globals=globals()))
```

```
31.858522263000168
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Examples: Simple Network

```
from torch import nn as ptnn
model = ptnn.Sequential(
        ptnn.Linear( in_features: 784, out_features: 4096), ptnn.ReLU(),
        ptnn.Dropout(),
        ptnn.BatchNorm1d(4096),
        ptnn.Linear( in_features: 4096, out_features: 2048), ptnn.ReLU(),
        ptnn.Dropout(),
        ptnn.BatchNorm1d(2048),
        ptnn.Linear( in_features: 2048, out_features: 1024), ptnn.ReLU(),
        ptnn.Dropout(),
        ptnn.BatchNorm1d(1024),
        ptnn.Linear( in_features: 1024, out_features: 512), ptnn.ReLU(),
        ptnn.Dropout(),
        ptnn.BatchNorm1d(512),
        ptnn.Linear( in_features: 512, out_features: 256), ptnn.ReLU(),
        ptnn.Dropout(),
        ptnn.BatchNorm1d(256),
        ptnn.Linear( in_features: 256, out_features: 10), ptnn.ReLU(),
        ptnn.LogSoftmax(dim=1)).to("cuda:0")
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Examples: Simple Network

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Making Our Own Binding

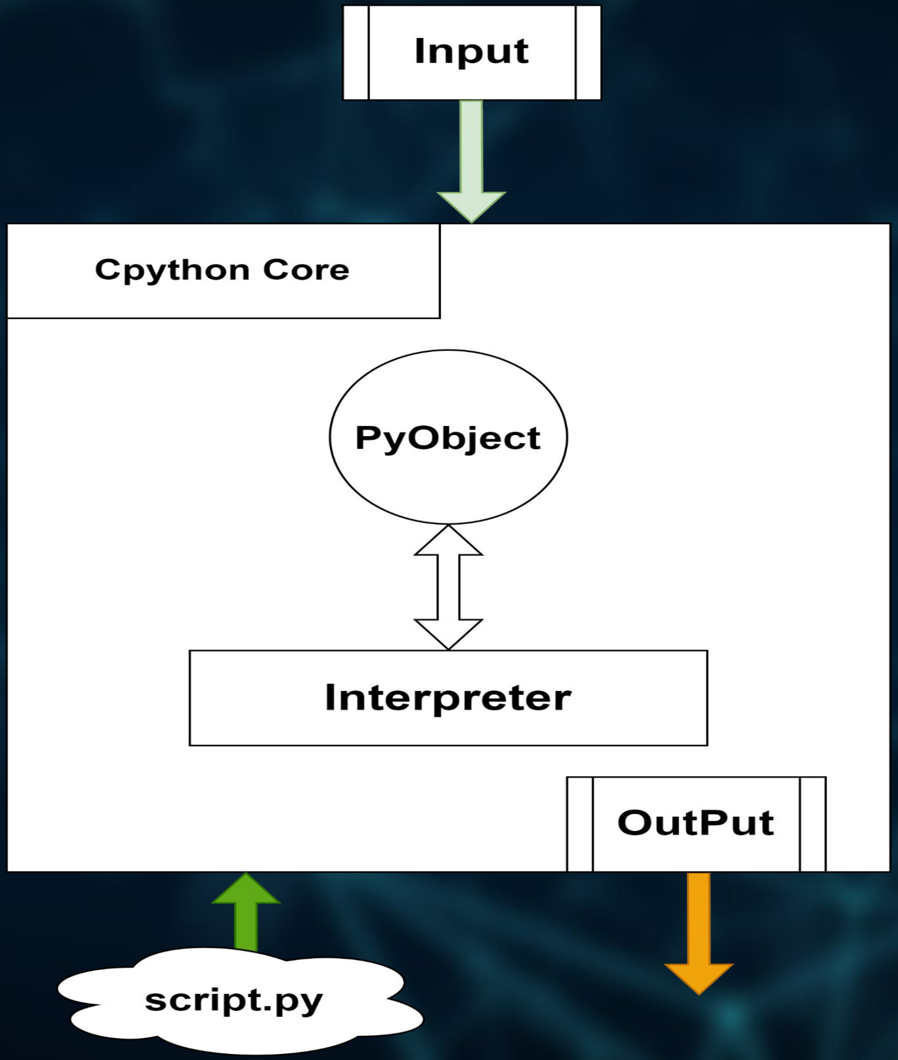
Sometimes we need our own functionality

```
float foo(int a, int b){
    return (2*a + 3*b) * 0.5;
}
```

CPython

- The Python Interpreter
  - One of the references and the most used one
  - its a C program that interprets the python programming language and runs it
  - Maintains the internal state of the python program
  - Everything is a PyObject

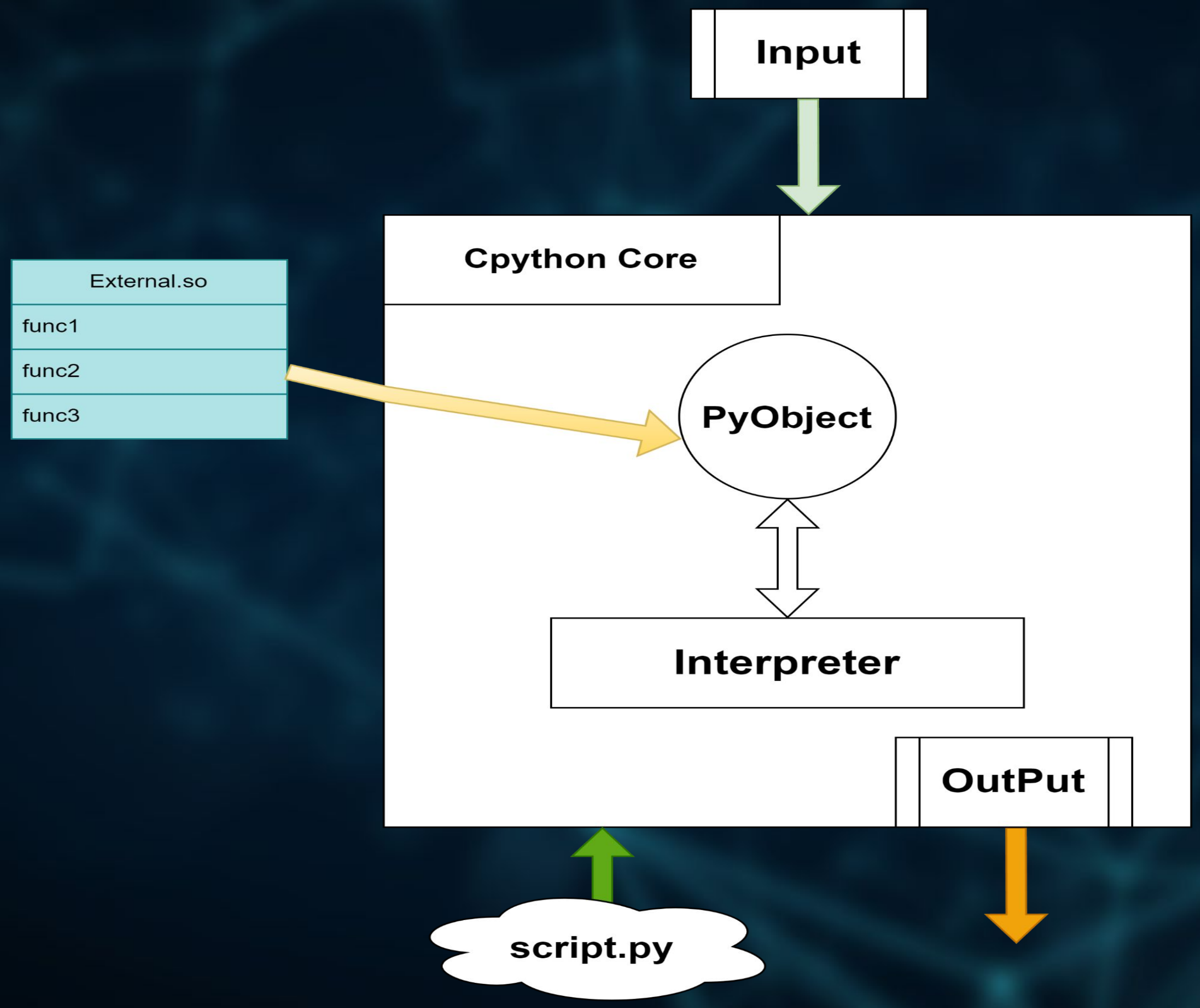
CPython



CPython: Extension Model

- CPython allows us to extend the language
  - Provides Python.h file that contains the API
  - Allows us to add hooks to be called from python

CPython



Input
Cpython Core
External.so
func1
func2
func3
PyObject
Interpreter
OutPut
script.py

CPython: Extension Model

```c
#include <Python.h>


// C function
float foo(int a, int b) {
    return (2 * a + 3 * b) * 0.5;
}
```

CPython: Extension Model

```c
// CPython wrapper for foo
static PyObject* py_foo(PyObject* self, PyObject* args) {
    int a, b;
    // Parse the Python arguments (expecting two integers)
    if (!PyArg_ParseTuple(args, "ii", &a, &b)) {
        return NULL;  // If parsing fails, return NULL
    }

    // Call the actual C function
    float result = foo(a, b);

    // Return the result as a Python float object
    return PyFloat_FromDouble(result);
}
```

CPython: Extension Model

```c
// Method definitions
static PyMethodDef FooMethods[] = {
    {"foo", py_foo, METH_VARARGS, "Calculate (2*a + 3*b) * 0.5"},
    {NULL, NULL, 0, NULL}  // Sentinel
};
```

CPython: Extension Model

```c
// Module definition
static struct PyModuleDef foomodule = {
    PyModuleDef_HEAD_INIT,
    "foo_module",  // Name of the module
    "A module that provides foo()",  // Module documentation
    -1,  // Size of per-interpreter state or -1 if global
    FooMethods  // Methods in the module
};
```

CPython: Extension Model

```c
// Module initialization function
PyMODINIT_FUNC PyInit_foo_module(void) {
    return PyModule_Create(&foomodule);
}
```

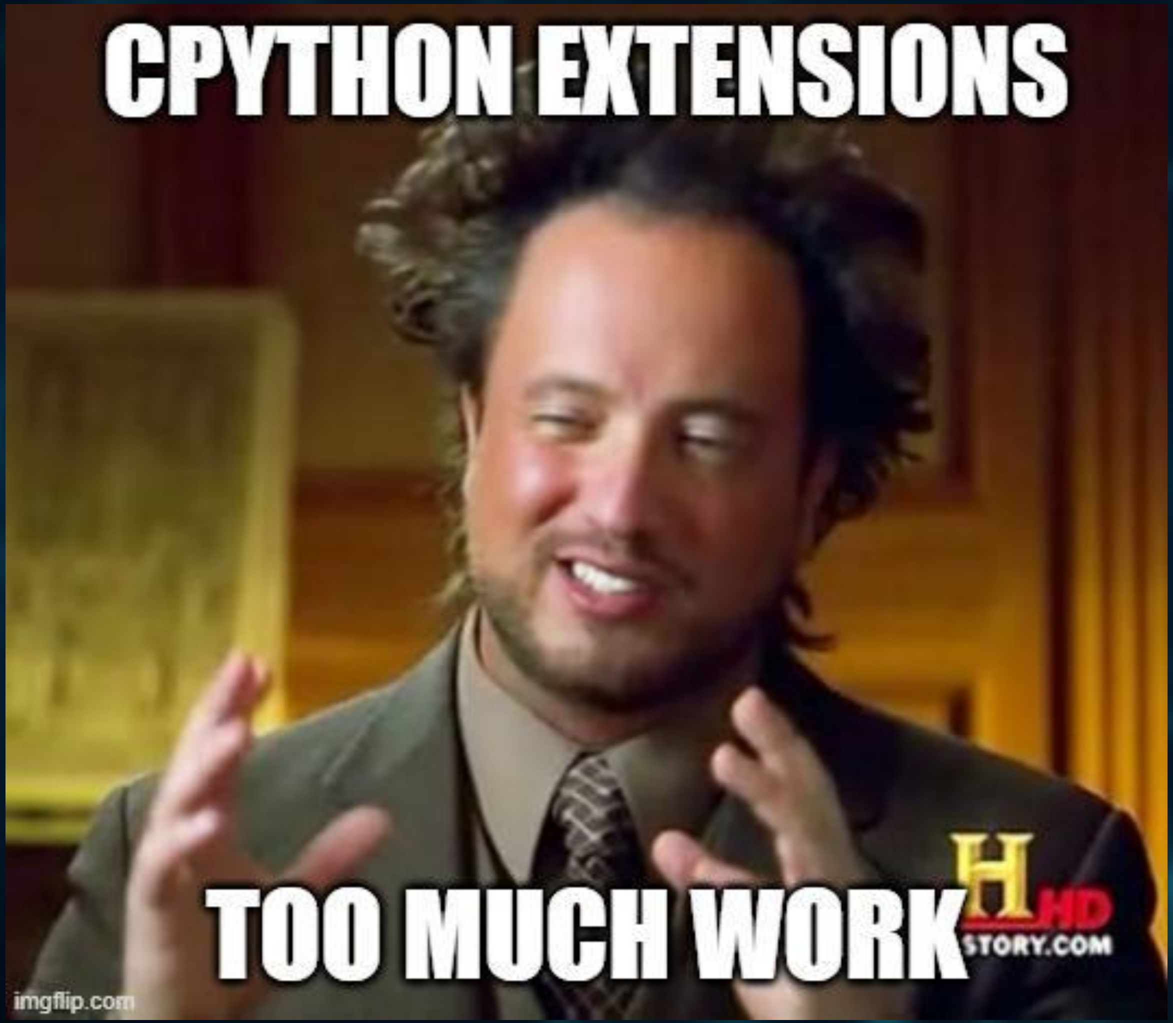
CPython: Extension Model

```python
from setuptools import setup, Extension

# Define the extension module
module = Extension("foo_module", sources=["foo_module.c"])

# Setup function
setup(
    name="foo_module",
    version="1.0",
    description="Python interface for the foo C function",
    ext_modules=[module],
)
```

CPython: Extension Model



Binding Tools

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Binding Tools

There are many Binding tools , during this talk we would focus on

- ❑ Static Binding from C++ to Python
  - ❑ PyBind11
  - ❑ Boost Python
- ❑ Dynamic Binding from Cpp to Python
  - ❑ CPPyy
- ❑ Static Binding language from C++ to Python and from Python to C++
  - ❑ Cython

Binding Tools: PyBind11

```cpp
#include <pybind11/pybind11.h>

namespace py = pybind11;

// The C++ function
float foo(int a, int b) {
    return (2 * a + 3 * b) * 0.5;
}

// Module definition
PYBIND11_MODULE(foo_module, m) {
    // Optional module docstring
    m.doc() = "A module that provides the foo function";

    // Expose the foo function
    m.def("foo", &foo, "A function that calculates (2*a + 3*b) * 0.5",
            // Optional argument names for better Python usability
            py::arg("a"), py::arg("b"));
}
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Binding Tools: PyBind11

```cmake
cmake_minimum_required(VERSION 3.29)
project(Bindings)

set(CMAKE_CXX_STANDARD 26)
find_package(pybind11 REQUIRED)

add_library(foomod SHARED main.cpp)
target_link_libraries(foomod pybind11_all_do_not_use)
```

Binding Tools: PyBind11

```python
from setuptools import setup

setup(
    name="foo_module",
    version="1.0",
    description="Python interface for the foo C++ function using pybind11",
)
```

Binding Tools: PyBind11

```
install(CODE "execute_process(
        WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
        COMMAND ${Python_EXECUTABLE}
        ${CMAKE_CURRENT_LIST_DIR}/setup.py install")
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Binding Tools: Boost Python

```cpp
#include <boost/python.hpp>

// C++ function
float foo(int a, int b) {
    return (2 * a + 3 * b) * 0.5;
}

// Module definition
BOOST_PYTHON_MODULE(foo_module) {
    using bp = boost::python;

    // Expose the C++ function 'foo' to Python
    bp.def("foo", foo, "A function that calculates (2*a + 3*b) * 0.5");
}
```

Binding Tools: Boost Python

```
cmake_minimum_required(VERSION 3.29)
project(Bindings)

set(CMAKE_CXX_STANDARD 26)
find_package(Boost COMPONENTS python3 REQUIRED)
find_package(PythonInterp 3 REQUIRED)
find_package(PythonLibs 3 REQUIRED)


add_library(foomod SHARED main.cpp)
target_link_libraries(foomod Boost::python3)
```

Binding Tools: What about classes?

```cpp
// C++ class definition
struct cpp_class {
    float foo(int a, int b) {
        return (2 * a + 3 * b) * 0.5;
    }

    float bar(int a, int b) {
        return (a + b) * 0.5;
    }
};
```

Binding Tools: What about classes?

```cpp
// Exposing cpp_class to Python using pybind11
PYBIND11_MODULE(cpp_class_module, m) {
    py::class_<cpp_class>(m, "cpp_class")
        .def(py::init<>())  // Expose the default constructor
        .def("foo", &cpp_class::foo, "A function that calculates (2*a + 3*b) * 0.5")
        .def("bar", &cpp_class::bar, "A function that calculates (a + b) * 0.5");
}
```

What About Performance

Binding Packs: But (Performance)

Platform	Time (second) [10 million runs]
C++	0.0042 (4.24 mili)
Python	1.2206 (1220.60 mili)
CPython	1.1058 (1105.80 mili)
PyBind11	5.74263 (5742.63 mili)

Binding Packs: But (Performance)

We have overheads

- ❑ The data needs to go back and forth from C++ object to PyObject
- ❑ Different Libraries use different methods
  - ❑ PyBind11 allocates almost everything as smart pointers or vectors and it makes it a bit slower
- ❑ Just like with CUDA we need to select the right parts to be optimised.

Binding For Better Performance

- Make Batch functions (Hot areas of code)
- Use Bindings objects for C++ conversion (py::dict)
- Use built in optimised type bindings like (py::array_t<double>)
- Return values: important to understand the lifetime of the returned object
- Use smart pointers instead of regular pointers

Binding Packs: Better Function for python run

```
float foo(py::array_t<int> a) {
    float sum = 0;
    float v = 2;
    const auto ptr = static_cast<const int*>(a.request().ptr);
    for (int i = 0; i < a.size(); i++) {
        sum += v++ * static_cast<float>(ptr[i]);
    }
```

Binding Packs: But (Performance)

Platform	Time (second) [10 million runs, 15 values]
C++	0.0624 (62.4 mili)
PyBind11	1.64263 (1642.63 mili)

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Cython

Cython

- PyBind11 and BoostPython are C++ language extensions
  - Can run only C++ code in python
- Cython
  - Rich language that looks like python
  - More features and much more flexible
  - We can run Python code from C++

C++ -> Python

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Cython: It All Starts from CPP class

```cpp
namespace important{
    struct abstract{
        virtual float foo(int a, int b) = 0;
        virtual float bar(int a, int b) = 0;
        virtual ~abstract() = default;
    };

    struct real : public abstract{
        float foo(int a, int b) override {
            return (2*a+3*b)*0.5;
        }

        float bar(int a, int b) override {
            return (a+b)*0.5;
        }
    };
}
```

Cython: PXD File

```
for libcpp.memory cimport shared_ptr

cdef extern from "class.hpp" namespace "important":
    cdef cppclass Cabstract "important::abstract":
        abstract() except+
        float foo(int a, int b)
        float bar(int a, int b)


    cdef cppclass Creal(Cabstract):
        real() except+
        float foo(int a, int b)
        float bar(int a, int b)
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Cython: PXD File

```
cdef class PyReal:
    cdef shared_ptr[Creal] c_real
```

Cython: PYX File

```
from my_module cimport *
from libcpp.memory cimport make_shared, nullptr
from cython.operator cimport dereference

cdef class PyReal:
    def __cinit__(self):
        c_real = make_shared[CReal]();

    def __init__(self):
        pass

    cpdef foo(int a, int b):
        dereference(c_real.get()).foo(a, b)
```

Cython: Cythonization

```
cython -3 --cplus my_model.pyx
```

Cython: Cythonization Cmake Version

```cmake
cmake_minimum_required(VERSION 3.29)
project(Bindings)

find_package(Python 3 REQUIRED COMPONENTS Interpreter Development)

add_custom_command(
    OUTPUT
        ${CMAKE_CURRENT_BINARY_DIR}/my_model.cpp
    DEPENDS
        ${CMAKE_CURRENT_BINARY_DIR}/my_model.pyx
        ${CMAKE_CURRENT_BINARY_DIR}/my_model.pxd

    COMMAND cython -3 --cplus ${CMAKE_CURRENT_BINARY_DIR}/my_model.pyx
    )

    add_library(my_model SHARED my_model.cpp)

    #we want it to be without lib....
    set_target_properties(py_my_modeltypes PROPERTIES PREFIX "")

    python_install_function(...)
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Python -> C++

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Cython: Public Functions

```
cdef public string load_json(string path):
    j = json.load(path)
    j['x'] = 'PYTHON'
    return j.dumps()
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Cython: C++ Init Python

```cpp
void PythonAutoLoader() {
  const char* libpython_path = "python so file";
  python_lib_handle_ = dlopen(libpython_path, RTLD_NOW | RTLD_GLOBAL);
  if (nullptr == python_lib_handle_) {
    throw std::runtime_error("Cannot load libpython");
  }

  PyImport_AppendInittab("my_model", PyInit_my_model);
  Py_Initialize();
  if (!PyImport_ImportModule("my_model")) {
    PyErr_Print();
    throw std::runtime_error("Cannot import my_model python interop module");
  }
}
```

Few Words About Exceptions

Cython: Exceptions

- We cannot propagate exceptions from python to C++
- It's easy to create a function in C++ that is invoked from cython

```
cdef extern from "Thrower.h" namespace "utils":
    cdef void ThrowCppRuntimeError(string message)

def throw_cpp_error(msg: string):
    ThrowCppRuntimeError(msg.encode("utf8"))
```

Dynamic Binding

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Dynamic Binding: Cppyy

- PyBind11, BoostPython and Cython are static
  - need to be compiled into a module
- Cppyy is dynamic binding
  - write C++ directly into python
  - supports CPython and PyPy (much faster on PyPy)
- Easy to use functionality

Dynamic Binding: Cppyy

```
cppyy.cppdef(r"""\
                 float foo(int a, int b) {
                  return (2*a+3*b)*0.5; } """)
cppyy.gbl.foo(10, 20)
40.0
```

Dynamic Binding: Cppyy

Platform	Time (second) [10 million runs, 15 values]
C++	0.0042 (4.24 mili)
Cppyy	0.7243 (724.3 mili)

Dynamic Binding: Cppyy

- Uses Cling/ clang base interpreter for C++
  - No pre install or compile
  - No need to duplicate STL
  - Full support for templates
  - Full support of inheritance
  - Full support for callbacks and lambdas

Cppyy : Examples

```
struct abstract {
    virtual int foo(int a, int b) = 0;
    virtual std::array<int, 10> bar(int n) = 0;
    virtual ~abstract() = default;
};

struct real : public abstract {
    real() = default;

    int foo(int a, int b) override {
        return static_cast<int>((2*a + 3*b)*0.5);
    }
    std::array<int, 10> bar(int n) override {
        return std::array{1*n, 2, 3, 4, 5, 6, 7, 8, 9, 10*n};
    }
};
```

Cppyy : Examples (including)

```
>>> import cppyy
>>> cppyy.include('features.h')
True
>>> from cppyy.gbl import real
>>> real
<class cppyy.gbl.real at 0x40d2b00>
```

Cppy : Examples (Calling C++ Class Method)

```
>>> r = real()
>>> r.foo(10, 20)
```

Cppyy : Examples (Cross Inheritance)

```
>>> cppyy.include('array')
True
>>> class PyReal(cppyy.gbl.abstract):
...     def foo(self, a, b):
...             return a+b+10
...     def bar(self, a):
...             return cppyy.gbl.array[int, 10]()
...
>>> pr = PyReal()
>>> pr.foo(10, 20)
40
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Cppy : Examples (Template Functions)

```cpp
template <typename R, typename... U, typename... Args>
R callit(R(*f)(U...), Args&&... args) {
    return std::invoke(f, std::forward<Args>(args)...);
}
```

Cppyy : Examples (Template Functions)

```
>>> from cppyy.gbl import callit
>>> def f(a: int, b: int) -> float:
...     return (2*a+3*b)*0.5
...
>>> callit(f, 10, 20)
40.0
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Closing Notes

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Dynamic Binding: Cppyy

- Use the right tools for your needs
- Don't implement everything by yourself
  - Be “lazy” and use open source
  - Try cythonizing pure python code to get performance
- If you have to write something by yourself
  - Be familiar with the rules of the binding pack your using
  - Understand how the transfer of data works
  - Avoid dangling pointers
  - Be aware of Threading problems in Python

QUESTIONS

?

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

THANK YOU FOR LISTENING

ALEX DATHSKOVSKY

ALEX.DATHSKOVSKY@SPEEDATA.IO

WWW.LINKEDIN.COM/IN/ALEXDATHSKOVSKY